Abstraction of Problems

- Data: abstracted as a word in a given alphabet.
  - \( \Sigma \): alphabet, a finite, non-empty set of symbols.
  - \( \Sigma^* \): all the words of finite length built up using \( \Sigma \).
- Conditions: abstracted as a set of words, called language.
  - Any subset \( L \subseteq \Sigma^* \) is a formal language.
- Unknown: Implicitly a Boolean variable: true if a word is the language; no, otherwise.
  - Given \( w \in \Sigma^* \) and \( L \subseteq \Sigma^* \), does \( w \in L \)?

Regular Languages

Hantao Zhang
http://www.cs.uiowa.edu/~hzhang/c135

The University of Iowa
Department of Computer Science

Formal Definition of Finite Automata

A finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where:
- \(Q\) is a finite set called the states
- \(\Sigma\) is a finite set called the alphabet
- \(\delta : Q \times \Sigma \to Q\) is the transition function
- \(q_0 \in Q\) is the start state also called initial state
- \(F \subseteq Q\) is the set of accept states, also called the final states

Finite Automata

- The simplest computational model is called a finite state machine or a finite automaton
- Representations:
  - Graphical
  - Tabular
  - Mathematical.
Computation of a Finite Automaton

- The automaton receives the input symbols one by one from left to right.
- After reading each symbol, $M_1$ moves from one state to another along the transition that has that symbol as its label.
- When $M_1$ reads the last symbol of the input it produces the output: accept if $M_1$ is in an accept state, or reject if $M_1$ is not in an accept state.

Applications

- Finite automata are popular in parser construction of compilers.
- Finite automata and their probabilistic counterparts, Markov chain, are useful tools for pattern recognition used in speech processing and optical character recognition.
- Markov chains have been used to model and predict price changes in financial applications.

Formal Definition of Acceptance

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and $w = a_1a_2 \ldots a_n$ be a string over $\Sigma$.

Then $M$ accepts $w$ if a sequence of states $r_0, r_1, \ldots, r_n$ exist in $Q$ such that the following hold:

1. $r_0 = q_0$
2. $\delta(r_i, a_{i+1}) = r_{i+1}$ for $i = 0, 1, \ldots, n - 1$
3. $r_n \in F$

Condition (1) says where the machine starts.
Condition (2) says that the machine goes from state to state according to its transition function $\delta$.
Condition (3) says when the machine accepts its input: if it ends up in an accept state.

Language Recognized by an Automaton

- If $L$ is the set of all strings that a finite automaton $M$ accepts, we say that $L$ is the language of the machine $M$ and write $L(M) = L$.
- An automaton may accept several strings, but it always recognizes only one language.
- If a machine accepts no strings, it still recognizes one language, namely the empty language $\emptyset$. 
Designing Finite Automata

- Whether it be of automaton or artwork, design is a creative process. Consequently it cannot be reduced to a simple recipe or formula.
- The approach:
  - Identify the finite pieces of information you need to solve the problem. These are the states.
  - Identify the condition (alphabet) to change from one state to another.
  - Identify the start and final states.
  - Add missing transitions.

Regular Languages

We say that a finite automaton $M$ recognizes the language $L$ if $L = \{w | M$ accepts $w\}$.

Definition A language is called regular language if there exists a finite automaton that recognizes it.

Regular Expressions

- Three base cases:
  - $\emptyset$ is a regular expression denoting the language $\emptyset$;
  - $\epsilon$ is a regular expression denoting the language $\{\epsilon\}$;
  - For any $a \in \Sigma$, $a$ is a regular expression denoting the language $\{a\}$;
- Three recursive cases: If $r_1$ and $r_2$ are regular expressions denoting languages $L_1$ and $L_2$, respectively, then
  - Union: $r_1 \cup r_2$ denotes $L_1 \cup L_2$;
  - Concatenation: $r_1 r_2$ denotes $L_1 L_2$;
  - Star: $r_1^*$ denotes $L_1^*$.

Operations on Regular Languages

Let $A$ and $B$ be languages. We define regular operations union, concatenation, and star as follows:
- Union: $A \cup B = \{x | x \in A \lor x \in B\}$
- Concatenation: $A \circ B = \{xy | x \in A \land y \in B\}$
- Star: $A^* = \{x_1 x_2 \ldots x_k | k \geq 0 \land x_i \in A, 0 \leq i \leq k\}$
  - Note:
    1. Because “any number” includes 0, $\epsilon \in A^*$, no matter what $A$ is.
    2. $A^+$ denotes $A \circ A^*$. 
Closure under Complementation

- **Theorem** That class of regular languages is closed under complementation.
- **Proof** For that we will first show that if $M$ is a DFA that recognizes a language $B$, swapping the accept and non-accept states in $M$ yields a new DFA that recognizes the complement of $B$.

Closure under Intersection

- **Theorem** That class of regular languages is closed under intersection.
- **Proof** Use cross-product construction of states.

Two ways to Introduce Nondeterminism

- More choices for the next state: Zero, one, or many arrows may exit from each state.
- A state may change to the next state without spending an input symbol: $\epsilon$-transitions.

Nondeterminism

- So far in our discussion, every step of a finite automaton computation follows in a unique way from the preceding step.
- We call this a **deterministic computation**. In a **nondeterministic** computation, choices may exist for the next state at any point.
- Nondeterminism is a generalization of determinism; hence, every finite automaton is a nondeterministic finite automaton (NFA).
Tree Computation of NFA

A way to think of a nondeterministic computation is as a tree of possibilities.
- The root of the tree corresponds to the start of the computation.
- Every branching point in the tree corresponds to a point in the computation at which the machine has multiple choices.
- The machine accepts if at least one of the computation branches ends in an accept state.

Formal Definition of NFA

An NFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where:
- \(Q\) is a finite set called the **states**.
- \(\Sigma\) is a finite set called the **alphabet**.
- \(\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)\)
  where \(\mathcal{P}(Q)\) is the power set of \(Q\).
- \(q_0 \in Q\) is the **start state** also called initial state.
- \(F \subseteq Q\) is the set of **accept states**, also called the final states.

In a DFA transition function is \(\delta : Q \times \Sigma \to Q\).

**notation:** For any alphabet \(\Sigma, \Sigma_\epsilon = \Sigma \cup \{\epsilon\}\)

Computation by an NFA

Let \(N = (Q, \Sigma, \delta, q_0, F)\) be an NFA and \(w\) a string over \(\Sigma\). We say that \(N\) accepts \(w\) if \(w = a_1 a_2 \ldots a_m, a_i \in \Sigma_\epsilon, 1 \leq i \leq m\), and a sequence of states \(r_0, r_1, \ldots, r_m\) exists in \(Q\) such that:
1. \(r_0 = q_0\)
2. \(r_{i+1} \in \delta(r_i, a_{i+1}), \text{ for } i = 0, 1, \ldots, m - 1\)
3. \(r_m \in F\)

Condition 1 says the machine's starting state.
Condition 2 says that state \(r_{i+1}\) is one of the allowable new states when \(N\) is in state \(r_i\) and reads \(a_{i+1}\). Note that \(\delta(r_i, a_{i+1})\) is a set.
Condition 3 says that the machine accepts the input if the last state is in the accept state set.

Why NFA?

- Constructing NFA is sometimes easier than constructing DFA.
- An NFA may be much smaller than a DFA that performs the same task.
- Computation of NFA is usually more expensive than that of DFA.
- Every NFA can be converted into an equivalent DFA.
- NFA provides good introduction to nondeterminism in more powerful computational models.
Application of NFA

Now we use the NFA to show that collection of regular languages is closed under regular operations union, concatenation, and star.

Theorem 1.25, 1.45 The class of regular languages is closed under the union operation.

Theorem 1.26, 1.47 The class of regular languages is closed under concatenation operation.

Theorem 1.49 The class of regular languages is closed under star operation.

Equivalence of NFA and DFA

Theorem Every NFA has an equivalent DFA to recognize the same language.

DFAs and NFAs recognize the same class of languages

This equivalence is both surprising and useful.

It is surprising because NFAs appears to have more power than DFA, so we might expect that NFA recognizes more languages

It is useful because describing an NFA for a given language sometimes is much easier than describing a DFA

The Formal Proof
Let \( N = (Q, \Sigma, \delta, q_0, F) \) be the NFA recognizing the language \( A \). We construct the DFA \( M \) recognizing \( A \).

Before doing the full construction, consider first the easier case when \( N \) has no \( \epsilon \) transitions.

Question
Is the class of languages recognized by NFAs closed under complementation?
Corollary 1.40

A language is regular iff some NFA recognizes it

Proof:
- If a language $A$ is recognized by an NFA then $A$ is recognized by the DFA equivalent, hence, $A$ is regular.

Theorem 1.54

A language is regular iff some regular expression describes it

Proof:
- If a language $A$ is described by a regular expression then $A$ is recognized by an NFA, hence, $A$ is regular.

Equivalence of NFA and DFA

The Formal Proof
Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing the language $A$. We construct the DFA $M$ recognizing $A$.

- Before doing the full construction, consider first the easier case when $N$ has no $\epsilon$ transitions.
- Then we consider the $\epsilon$ transitions.

Notation: For any $R \subseteq Q$ define $E(R)$ to be the collection of states that can be reached from $R$ by going only along $\epsilon$ transitions, including the members of $R$ themselves. Formally:
$$ E(R) = R \cup \{ q \in Q \mid \exists r_1 \in R, r_2, \ldots, r_k \in Q, r_{i+1} = \delta(r_i, \epsilon), r_k = q \} $$

Corollary 1.40

A language is regular iff some NFA recognizes it

Proof:
- If a language $A$ is recognized by an NFA then $A$ is recognized by the DFA equivalent, hence, $A$ is regular.
- If a language $A$ is regular, it means that it is recognized by a DFA. But any DFA is also an NFA, hence, the language is recognized by an NFA.
Theorem 1.54

A language is regular iff some regular expression describes it.

Proof:

- If a language $A$ is described by a regular expression then $A$ is recognized by an NFA, hence, $A$ is regular.
- (sketch) If a language $A$ is regular, it means that it is recognized by a DFA. Then we can always deduce a regular expression from it.

Nonregular languages

Consider the language $B = \{0^n1^n | n \geq 0\}$.

- If we attempt to find a DFA that recognizes $B$ we discover that such a machine needs to remember how many 0s have been seen so far as it reads the input.
- Because the number of 0s isn’t limited, the machine needs to keep track of an unlimited number of possibilities.
- This cannot be done with any finite number of states.

Language nonregularity

- The technique for proving nonregularity of some languages is provided by a theorem about regular languages called pumping lemma.
- Pumping lemma states that all regular languages have a special property.
- If we can show that a language does not have this property we are guaranteed that it is not regular.

Intuition may fail us

- Just because a language appears to require unbounded memory in order to be recognized, it doesn’t mean that it is necessarily so.
- Example:
  - $C = \{w | w$ has an equal number of 0s and 1s $\}$ is not regular.
  - $D = \{w | w$ has an equal number of 01 and 10 as substrings $\}$ is regular.
**Theorem 1.37**

**Pumping Lemma:** If $A$ is a regular language, then there is a number $p$ (the pumping length) where, if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$
2. $|y| > 0$
3. $|xy| \leq p$

---

**Pumping property**

*All strings in the language can be “pumped” if they are at least as long as a certain special value, called the pumping length.*

**Meaning:** each such string in the language contains a section that can be repeated any number of times with the resulting string remaining in the language.

---

**Pumping lemma’s proof**

Let $M = (Q, \Sigma, \delta, q_1, F)$ by a DFA recognizing $A$ and $p$ be the number of states of $M$; let $s = s_1s_2\ldots s_n$ be a string over $\Sigma$ with $n \geq p$ and $r_1, r_2, \ldots, r_{n+1}$ be the sequence of states while processing $s$, i.e., $r_{i+1} = \delta(r_i, s_i), 1 \leq i \leq n$.

1. $n + 1 \geq p + 1$ and among the first $p + 1$ elements in $r_1, r_2, \ldots, r_{n+1}$ two must be the same state, say $r_j, r_k$.
2. Because $r_k$ occurs among the first $p + 1$ places in the sequence starting at $r_1$, we have $k \leq p + 1$
3. Now let $x = s_1 \ldots s_j - 1, y = s_j \ldots s_{k-1}, z = s_k \ldots s_n$.

---

**Proof idea**

Let $M = (Q, \Sigma, \delta, q_1, F)$ be a DFA that recognizes $A$

1. Assign a pumping length $p$ to be the number of states of $M$
2. Show that any string $s \in A$, $|s| \geq p$ may be broken into three pieces $xyz$ satisfying the pumping lemma’s conditions
3. If there are no strings in $A$ of length at least $p$ then theorem becomes vacuously true because all three conditions hold for all strings of length at least $p$ if there are no such strings
Using pumping lemma

Prove that a language $A$ is not regular using pumping lemma:

1. Assume that $A$ is regular in order to obtain a contradiction
2. The pumping lemma guarantees the existence of a pumping length $p$ such that all strings of length $p$ or greater in $A$ can be pumped
3. Find $s \in A$, $|s| \geq p$, that cannot be pumped: demonstrate that $s$ cannot be pumped by considering all ways of dividing $s$ into $x, y, z$, showing that for each division one of the pumping lemma conditions, (1) $xy^iz \not\in B$, (2) $|y| > 0$, (3) $|xy| \leq p$, fails.

Example, continuation

Consider the cases:

1. $y$ consists of 0s only. In this case $xyyz$ has more 0s than 1s and so it is not in $B$, violating condition 1
2. $y$ consists of 1s only. This leads to the same contradiction
3. $y$ consists of 0s and 1s. In this case $xyyz$ may have the same number of 0s and 1s but they are out of order with some 1s before some 0s hence it cannot be in $B$ either

The contradiction is unavoidable if we make the assumption that $B$ is regular so $B$ is not regular

Note

- As $x$ takes $M$ from $r_1$ to $r_j$, $y$ takes $M$ from $r_j$ to $r_j$, and $z$ takes $M$ from $r_j$ to $r_{n+1}$, which is an accept state, $M$ must accept $xy^iz$, for $i \geq 0$
- We know that $j \neq k$, so $|y| > 0$;
- We also know that $k \leq p + 1$, so $|xy| \leq p$

Thus, all conditions are satisfied and lemma is proven

Applications

Example 1.38 prove that $B = \{0^n1^n|n \geq 0\}$ is not regular

Assume that $B$ is regular and let $p$ be the pumping length of $B$. Choose $s = 0^p1^p \in B$; obviously $|0^p1^p| > p$. By pumping lemma $s = xyz$ such that for any $i \geq 0$, $xy^iz \in B$
**Example 1.39**

Prove that $C = \{w | w \text{ has an equal number of 0s and 1s}\}$ is not regular.

**Proof:** Assume that $C$ is regular and $p$ is its pumping length. Let $s = 0^p1^p$ with $s \in C$. Then pumping lemma guarantees that $s = xyz$, where $xy^iz \in C$ for any $i \geq 0$.

---

**Note**

If we take the division $x = z = \epsilon$, $y = 0^p1^p$ it seems that indeed, no contradiction occurs. However:

- Condition 3 states that $|xy| \leq p$, and in our case $xy = 0^p1^p$ and $|xy| > p$. Hence, $0^p1^p$ cannot be pumped.
- If $|xy| \leq p$ then $y$ must consists of only 0s, so $xyyz \not\in C$, because there are more 0-s than 1-s. This gives us the desired contradiction.

---

**An alternative method**

Use the fact that $B$ is nonregular.

- If $C$ were regular then $C \cap 0^*1^*$ would also be regular because $0^*1^*$ is regular and intersection of regular languages is a regular language.
- But $C \cap 0^*1^* = B$ which is not regular.
- Hence, $C$ is not regular either.

---

**Other selections**

Selecting $s = (01)^p$ leads us to trouble because this string can be pumped by the division: $x = \epsilon$, $y = 01$, $z = (01)^{p-1}$. Then $xy^iz \in C$ for any $i \geq 0$. 
**Note**
- Condition 3 is again crucial because without it we could pump $s$ if we let $x = z = \epsilon$, so $xyz \in F$.
- Choosing $s = 0^p1^p1$, the string that exhibits the essence of the nonregularity of $F$. If we chose, say $0^p0^p \in F$ we fail because this string can be pumped.

**Example 1.40**
Show that $F = \{ww | w \in \{0,1\}^*\}$ is nonregular using pumping lemma.

**Proof:** Assume that $F$ is regular and $p$ is its pumping length. Consider $s = 0^p1^p1 \in F$. Since $|s| > p$ $s = xyz$ and satisfies the conditions of the pumping lemma.

**Searching for a contradiction**
The elements of $D$ are strings whose lengths are perfect squares. Looking at first perfect squares we observe that they are: 0, 1, 4, 9, 25, 36, 49, 64, 81, …
- Note the growing gap between these numbers: large members cannot be near each other
- Consider two strings $xy^i z$ and $xy^{i+1} z$ which differ from each other by a single repetition of $y$.
- If we chose $i$ very large the lengths of $xy^i z$ and $xy^{i+1} z$ cannot be both perfect square because they are too close to each other.

**Example 1.41**
Show that $D = \{1^n^2 | n \geq 0\}$ is nonregular.

**Proof:** by contradiction. Assume that $D$ is regular and let $p$ be its pumping length. Consider $s = 1^p^2 \in D$, $|s| \geq p$. Pumping lemma guarantees that $s$ can be split, $s = xyz$, where for all $i \geq 0$, $xy^i z \in D$. 
**Value of i for contradiction**

To calculate the value for $i$ that leads to contradiction we observe that:

- $|y| \leq |s| = p^2$
- Let $i = p^4$. Then $|y| \leq p^2 = \sqrt{p^3} < 2\sqrt{p^3} + 1 \leq 2\sqrt{|xy^iz|} + 1$

**Turning this idea into a proof**

Calculate the value of $i$ that gives us the contradiction.

- If $m = n^2$, calculating the difference we obtain $(n + 1)^2 - n^2 = 2n + 1 = 2\sqrt{m} + 1$
- By pumping lemma $|xy^iz|$ and $|xy^{i+1}z|$ are both perfect squares. But letting $|xy^iz| = m$ we can see that they cannot be both perfect square if $|y| < 2\sqrt{|xy^iz|} + 1$, because they would be too close together.

**Searching for a contradiction**

- Let $s = 0^{p+1}p$; From decomposition $s = xyz$, from condition 3, $|xy| \leq p$ it results that $y$ consists only of 0s.
- Let us examine $xyyz$ to see if it is in $E$. Adding an extra-copy of $y$ increases the $n$ numbers of zeros. Since $E$ contains all strings $0^*_1*$ that have more 0s than 1s will still give a string in $E$.

**Example 1.42**

Sometimes “pumping down” is useful when we apply pumping lemma.

- We illustrate this using pumping lemma to prove that $E = \{0^i1^i | i > j \}$ is not regular
- **Proof:** by contradiction using pumping lemma. Assume that $E$ is regular and its pumping length is $p$. 
Try something else

- Since \( xy^i z \in E \) even when \( i = 0 \), consider \( i = 0 \) and 
  \( xy^0 z = xz \in E \).
- This decreases the number of 0s in \( s \).
- Since \( s \) has just one more 0 than 1 and \( xz \) cannot have 
  more 0s than 1s,
  \((xyz = 0^{p+1}1p \text{ and } |y| \neq 0)\)
  \( xz \) cannot be in \( E \).

This is the required contradiction